Game of Drones: Módulo PSxDron de gestión de vuelo y de penalizaciones por impacto infrarrojo (IR)

Omar Raya Rodríguez

Resumen—Game of Drones es un modo de juego con drones donde dos equipos compiten por la conquista de bases distribuidas por el campo de juego. El objetivo es apoderarse del mayor número de bases, ya que al ocuparlas, estas otorgan puntos al poseedor, concediendo al equipo con mayor cantidad de puntos la victoria en la partida. El módulo PSxDron es el elemento que hace de vínculo entre un mando de PlayStation 3 (PS3), que actúa como controlador de vuelo, y una emisora radiocontrol que actúa simplemente como transmisor. Este módulo hace posible dar la orden de disparo, controlar de manera remota el vuelo del dron, y gestionar las diferentes penalizaciones posibles por impacto IR existentes en las normas del juego.

Paraules clau—Throttle, Roll, Yaw, Pitch, PPM, PWM, Arduino, USB_Host_Shield, PSxDron, DualShock 3, Bloque de control, Bloque de gestión, Bloque de simulación, Bloque de transmisión, Sistema, Implementación.

Abstract— Game of Drones is a game mode with drones where two teams compete for the conquest of a series of bases distributed on the playing field. The objective is to get hold of the largest number of bases, as they grant points to the possessor, granting the team with more points the victory in the game. The PSxDron module is the element that makes the link between a PlayStation 3 controller, which acts as a flight controller, and a radio control transmitter that acts simply as a transmitter. This module makes it possible both to give the firing order, as well as to remotely control the drone's flight, and to manage the different penalties available for IR impact according to the rules of the game.

Index Terms— Throttle, Roll, Yaw, Pitch, PPM, PWM, Arduino, USB_Host_Shield, PSxDron, DualShock 3, Control block, Management block, Simulation block, Transmission block, System, Implementation.

1 INTRODUCCIÓN

GAME of Drones es un modo de juego con drones y por equipos que consiste en la conquista de bases enemiga distribuidas por un campo de batalla para la obtención de puntos. La partida se construye sobre una batalla campal aérea donde los jugadores de ambos equipos ponen a prueba sus habilidades en el manejo de drones y su puntería disparando.

Este modo de juego mediante el uso del módulo PSxDron te da la oportunidad de pilotar una aeronave con capacidad de disparo multiefecto, ya que el modo de juego dispone de tres tipos de arma equipables. Los diferentes efectos de penalización están totalmente gestionados por este módulo.

2 OBJETIVOS

Este proyecto tiene como objetivo diseñar y desarrollar uno de los 4 bloques que componen el sistema de juego *Game of Drones,* y que se corresponde con la gestión de vuelo, disparo, y penalizaciones por impacto IR del cuadricóptero.

El módulo que se pretende diseñar e implementar será capaz por un lado de gestionar un mando de PlayStation 3 y convertir la señal que genera en una entendible por una emisora radiocontrol para drones, y por otro lado, de aplicar las diversas penalizaciones de movimiento producidas por el impacto del emisor IR incorporado en los drones, así como de dar la orden de disparo para accionar el emisor IR perteneciente al dron asociado al mando.

El sistema que se pretende conseguir empleará el mando de dron como transmisor de las señales de control producidas por el mando de PS3, las cuales tienen como objetivo accionar los diferentes motores con los que cuenta el cuadricóptero para llevar a cabo los diferentes movimientos en pleno vuelo.

Por otro lado, tanto la emisión de la orden de disparo como la recepción de la información acerca de las penalizaciones aplicables al movimiento del dron, se realizarán mediante un puerto serie que conecta el módulo con el ordenador central de gestión del juego.

Los requisitos funcionales del sistema son los que siguen:

- Tiempo de respuesta entre que se genera la orden de disparo y ésta se ejecuta, inferior o igual a 30 ms.
- Vuelo del dron suficientemente estable como para pilotarlo en un entorno virtual (simulador).
- Jugabilidad *user friendly*.

3 ESTADO DEL ARTE

Los drones son aeronaves que vuelan sin tripulación y que pueden ser controladas de manera remota o programadas para realizar un vuelo autónomo. En sus orígenes, estos vehículos tenían una aplicación puramente militar, y llevaban a cabo misiones tales como reconocimiento de objetivos o ataques estratégicos.

En la actualidad, el uso de drones está ampliamente extendido en la sociedad, utilizándose en ámbitos diversos tales como la extinción de fuegos, la video-vigilancia, el análisis del relieve y de las propiedades del suelo, el medio ambiente, el transporte y la entrega de mercancías o la distribución de señal wifi. Como podemos observar las aplicaciones son infinitas, pero el caso de uso en el que basa el proyecto es el uso recreativo, donde encontramos aplicaciones como las carreras de drones, donde se debe esquivar obstáculos o superar una ruta específica lo más rápido posible, o los combates aéreos. Estas aplicaciones, junto a la ya conocida pasión por los videojuegos que existe en nuestra sociedad, da forma tanto a la propuesta de juego que se pretende realizar mediante los diferentes módulos de *Game of Drones,* como concretamente al módulo que se detalla en apartados posteriores del informe, donde se trata de dar una solución alternativa a las existentes en el mercado para unificar la jugabilidad de los videojuegos, que se consigue con el mando de PlayStation, con el mundo de los drones recreativos.

A día de hoy existen en el mercado drones de combate capaces de conquistar bases y de disparar mediante IR, así como una gran variedad de mandos de control o aplicaciones móvil que permiten vincular el mando de PS3 al dron para su pilotaje. A pesar de ello, no se ha encontrado ninguna solución que incluya el uso de un microcontrolador para realizar dicha vinculación, siendo esto el objetivo del proyecto.

4 SISTEMA PROPUESTO

Observamos en la imagen 1 la arquitectura del sistema diseñado e implementado para el cumplimiento de los objetivos marcados en el proyecto. Este sistema está compuesto por 4 bloques diferenciados, los cuales se presentan en los apartados 4.1, 4.2, 4.3 y 4.4.

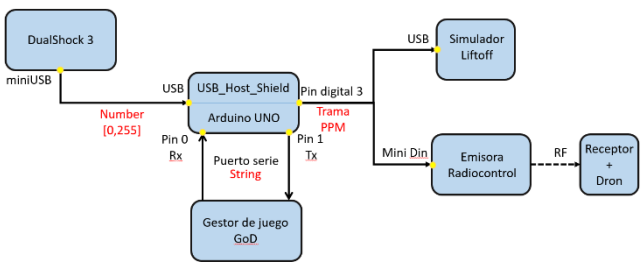

Imagen 1. Arquitectura del sistema

4.1 Bloque de control

El bloque de control está compuesto por un mando de PlayStation, concretamente el de la tercera generación que salió al mercado el 23 de marzo del 2007 en Europa, y que se denomina DualShock 3. Este mando revolucionó los controladores para consolas al añadir las funciones de detección de movimiento y vibración. El controlador es capaz de detectar el movimiento en 6 ejes mediante acelerómetros (3 posicionales en el espacio y 3 de rotación). A demás cuenta con conectividad Bluetooth y USB-miniUSB, con una batería de litio de aproximadamente 30 horas de autonomía, con botones analógicos capaces de detectar la presión con una resolución de 8 bits, y dos joysticks.

Este controlador es el empleado en el sistema PSxDron para realizar las órdenes de control de vuelo[1] que permiten dirigir el dron por el espacio. Para ello, se emplean los botones que se anuncian a continuación:

- **Botón Triangulo:** Empleado para cambiar de arma entre todas las disponibles en el juego *Game of Drones.*
- **Botón R1:** Empleado para llevar a cabo la acción de disparar. Genera una orden de disparo que se envía mediante un puerto serie al ordenador de gestión del juego *Game of Drones,* el cual transmitirá la orden al dron asociado para que accione el emisor IR.
- **Botón L1:** Empleado para activar o desactivar el modo "Control avanzado de vuelo". Este modo permite llevar a cabo movimientos avanzados mediante el accionamiento de varios botones simultáneamente a través de un único botón. En la versión actual del SW se ha desarrollado la funcionalidad de giro en ejes simultáneos, donde se unifican los movimientos de Yaw* y Roll*. La explicación detallada la encontramos en el aparatado 6.2.5.
- **Botón R2:** Empleado para controlar el throttle* del cuadricóptero. Este botón sensible a la presión aumenta o disminuye el throttle* del dron de manera directamente proporcional a la presión ejercida al accionarlo.
- **Joystick izquierdo:** Empleado para controlar el Yaw* derecho e izquierdo del dron.
- **Joystick derecho:** Empleado para controlar el roll* derecho e izquierdo del dron así como el pitch* delantero y trasero.

La gestión del DualShock 3 se lleva a cabo mediante una librería denominada USB_Host_Shield que incluye las funciones pertinentes para la obtención de los valores generados por los botones anteriormente comentados. Este controlador se conectará mediante un cable USB-miniUSB al bloque de gestión, donde se encuentra el SW de gestión desarrollado y la librería comentada.

[1] Los diferentes movimeintos que puede realizar un dron (*) se detallan en la sección 5.1

Las funciones incluidas en la librería USB_Host_Shield nos permiten obtener un conjunto de valores asociados a cada botón que se encuentran comprendidos entre los valores 0 y 255. Este valor, a través de una conversión explicada en detalle en la sección 6.2.6 determinará el ancho de pulso en la modulación de la señal PWM correspondiente a cada motor del cuadricóptero.

4.2 Bloque de gestión

El bloque de gestión está formado por dos dispositivos HW diferenciados. En primer lugar encontramos la Arduino USB_Host_Shield, y en segundo lugar la Arduino ONE.

4.2.1 Arduino USB_Host_Shield

La Arduino USB_Host_Shield es una placa complementaria a la Arduino ONE y que se basa en un controlador de periféricos MAX3421E. Este controlador contiene toda la lógica digital y los circuitos analógicos necesarios para conectar y controlar un periférico USB.

Esta placa se comunica con la Arduino ONE utilizando el bus SPI a través del encabezado ICSP mediante los pines digitales 10, 11, 12 y 13.

Este módulo permite conectar el DualShock 3, mediante la conexión USB-miniUSB, e interactuar con él a través de la Arduino ONE por medio de la librería anteriormente comentada.

4.2.2 Arduino ONE

Arduino es una plataforma de electrónica de código abierto que permite desarrollar HW y SW de manera rápida, y que cuenta con una comunidad activa de desarrolladores. Esta plataforma nació en el *Ivrea International Design Institute* como una herramienta de aceleración del prototipado de sistemas embebidos.

La Arduino ONE, una de las muchas placas disponibles, es una placa de desarrollo hardware comúnmente empleada para el prototipado de dispositivos destinados a la interacción tanto con el mundo real mediante sensores y actuadores, como con otros dispositivos periféricos.

Las especificaciones técnicas relevantes de la placa de prototipado son las siguientes:

- **Microprocesador:** ATmega328P a 16 MHz.
- **Pines digitales:** 14 de I/O, donde 6 de ellos pueden ser utilizados como salidas PWM.
- **Pines analógicos:** 6 de I/O.

Esta placa será la que contendrá el SW desarrollado, a través de la IDE de Arduino, para la gestión tanto del mando como de las penalizaciones, así como de la emisión de la orden de disparo. Observamos en la imagen 2 el diagrama de flujo de la ejecución del programa, el cual se explica a continuación.

4.2.2.1 Diagrama de Flujo de la ejecución del programa

1. **Setup:** En la fase de setup se realiza la inicialización de todas las variables necesarias a lo largo de la ejecución, así como la configuración de los pines (como entrada o salida), el puerto serie y el USB.
2. **Gestión del puerto serie:** En esta fase se realizan las 2 acciones siguientes:
    a. **Lectura del puerto serie** a través del pin digital 0.
    b. **Decodificación del mensaje** para la obtención de los tags indicadores del tipo de arma y de área afectada por el impacto. La estructura del mensaje es la correspondiente a la imagen 9.
3. **Gestión del tipo de arma:** En esta fase se gestiona la acción de cambio de arma realizable mediante el botón triangulo. Las armas disponibles son la de hielo (Tag ICE), la eléctrica (Tag ELE) y la nuclear (Tag NUC).
4. **Gestión del disparo:** En esta fase se gestiona la orden de disparo, enviando el mensaje de la imagen 12 por el puerto serie en caso de llevarse a cabo (pin digital 1).
5. **Gestión de movilidad avanzada:** En esta fase se gestiona el flag de activación/desactivación de la movilidad avanzada, permitiendo la ejecución simultánea de controles mediante el uso de un único control en caso de activarse. La explicación detallada la encontramos en el aparatado 6.2.5.
6. **Gestión de vuelo:** En esta fase se realiza la lectura del puerto USB para la obtención del valor de los diferentes botones relacionados con el control del dron. Obtenemos para cada control (R2 y Joysticks) un valor comprendido entre 0 y 255, y posteriormente se realiza una conversión map a un rango de valores de entre 1000 y 2000. El valor obtenido tras la conversión es el valor PWM que se transmitirá mediante una trama PPM. El proceso de transformación de PWM a PPM se explica en el apartado 6.2.8.
7. **Gestión de las penalizaciones:** En esta fase se aplican las penalizaciones correspondientes al tipo de arma y zona de impacto. Las penalizaciones existentes en función del tipo de arma son las siguientes:
    a. **Ralentización:** Reduce la movilidad del dron en los movimientos de Yaw, Roll y Pitch en función de la zona de impacto.
    b. **Incapacitación del arma:** Desactiva el arma del dron e imposibilita la acción de disparo.
    c. **Reducción de la potencia:** Reduce la potencia entregada a los motores del cuadricóptero.
8. **Creación de la trama PPM:** En esta fase se lleva a cabo la implementación de la trama PPM en función de los valores PWM obtenidos tras la gestión del vuelo y la aplicación de las penalizaciones. Esta trama es la encargada de transportar la información acerca de la potencia suministrada a cada uno de los motores del cuadricóptero, y es

enviada a través del pin digital 3.

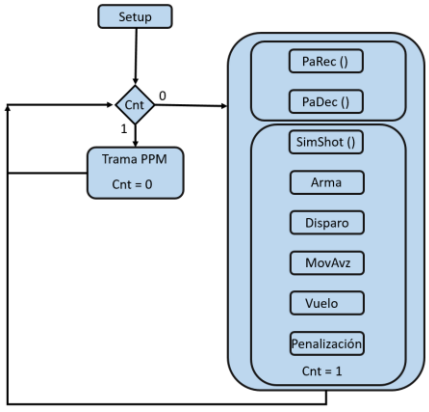

Imagen 2. Diagrama de flujo de la ejecución del programa

4.3 Bloque de simulación

Este bloque está compuesto por Liftoff, un simulador de vuelo desarrollado por LuGus Studios, ImmersionRC y Fat Shark, capaz de aportar una experiencia inmersiva y realista gracias a las físicas incorporadas en los drones, y a la posibilidad de utilizar gafas FPV junto a una gran variedad de aeronaves de todo tipo.

Este simulador está destinado a facilitar a los usuarios el aprendizaje y/o mejora de las habilidades de vuelo en un amplio conjunto de entornos, en los que se incluye una pista con puntos de control para carreras, por lo que es perfecto para testear la jugabilidad conseguida con el mando de PS3.

Por otro lado, el simulador permite utilizar una configuración de vuelo ACRO, donde la velocidad angular de la aeronave se controla de manera manual. A demás, permite utilizar un alto número de controladores remotos tales como un mando de PlayStation o una emisora radiocontrol, así como configurar tu propio mando a partir de una conexión con protocolo de comunicación PPM. Este hecho permite comparar la jugabilidad con controladores diferentes.

Este entorno virtual ha sido empleado para testear el correcto funcionamiento del módulo PSxDron en cada una de las fases de desarrollo. La conexión con el módulo de gestión se realiza mediante un conector al pin digital 3 (salida PPM) junto a un adaptador USB.

4.4 Bloque de transmisión y recepción

El bloque de transmisión y recepción está compuesto por dos componentes diferenciados y que se explican a continuación:

- **Emisora radiocontrol:** Se utiliza para la transmisión de la señal generada por el mando de PS3.
- **Dron:** Aeronave con el receptor vinculado al transmisor de la emisora radiocontrol.

Este bloque se conecta con el bloque de gestión mediante una conexión por cable desde el puerto de entrenamiento del mando del dron (conexión mini-DIN de 4 pines) hacía el pin digital 3 (salida PPM).

5 PROTOCOLOS DE COMUNICACIÓN

En el mundo de los drones radiocontrol existen variedad de protocolos de comunicación para la relación Tx-Rx, pero en este proyecto nos centraremos en los PWM y PPM. Antes de dar paso a la explicación de cada uno, debemos plantear el contexto haciendo una breve descripción de cuáles son las necesidades que presenta un dron en cuanto a la comunicación.

5.1 ¿Qué necesita un dron?

Situándonos en un modelo de dron básico donde lo único que es capaz de realizar es desplazamientos por el espacio (imagen 3), es fácil darse cuenta de que lo único que necesita saber del transmisor es que movimiento debe hacer.

Los movimientos que puede realizar un dron, que vienen determinados por la potencia suministrada a cada uno de los motores y el par que presentan, son los siguientes:

- **Throttle:** Este es un movimiento de aumento o disminución de altura. Todos los motores aumentan o disminuyen la velocidad de rotación de manera simultánea.
- **Pitch:** Este movimiento consiste en un desplazamiento en la dirección del dron:
  - **Delante:** Los motores traseros rotan a mayor velocidad que los delanteros.
  - **Atrás:** Los motores delanteros rotan a mayor velocidad que los traseros.
- **Roll:** Este movimiento consiste en un desplazamiento lateral:
  - **Derecha:** Los motores izquierdos rotan a mayor velocidad que los derechos.
  - **Izquierda:** Los motores derechos rotan a mayor velocidad que los izquierdos.
- **Yall:** Este es un movimiento de rotación:
  - **Derecha:** Dos motores diametralmente opuestos aumentan la velocidad de rotación respecto a los otros dos, y presentan un par a derecha.
  - **Izquierda:** Dos motores diametralmente opuestos aumentan la velocidad de rotación respecto a los otros dos, y presentan un par a izquierda.

Una vez conocemos los movimientos que puede realizar un dron y la forma de realizarlos mediante la aplicación de potencia y par a los motores, pasamos a explicar cómo el transmisor da las instrucciones mediante los protocolos de comunicación.

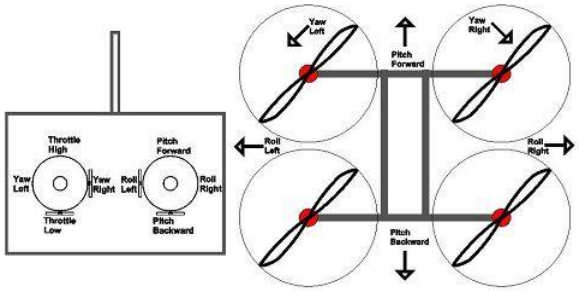

Imagen 3. Movimientos de un dron

5.1 PWM

La modulación por ancho de pulso (PWM) (imagen 4) de sus siglas en inglés (*Pulse-width modulation*) está formada por una señal periódica cuadrada con dos estados (high/low). Esta señal varia a lo largo del tiempo la relación entre el tiempo que está en alto (Th) y el tiempo que está en bajo (Tl), donde la suma de ambos tiempos es igual al periodo de la señal (T). En el caso de uso de los drones radiocontrol, la duración del pulso oscila entre los 1 ms y 2 ms, siendo cada uno de esos valores el máximo par en los motores hacia izquierda y hacia derecha, y 1.5 ms el punto de parada.

El objetivo de los cambios de estado entre Th y Tl es variar el ciclo de trabajo, que es la relación entre Th y T, para controlar la cantidad de energía que se envía a una carga (motor en nuestro caso). Este hecho determina que cada señal PWM se asocia a un motor.

La energía que se envía al motor será mayor cuanto más ciclo de trabajo tengamos.

Utilizando esta modulación podemos deducir que a mayor número de motores en nuestro dron, mayor deberá ser el número de canales de transmisión, y que en nuestro caso de uso (Cuadricóptero) el número de señales PWM será 4. El incremento del número de canales en el transmisor se traduce en un incremento en la complejidad del receptor y de la posterior gestión de las señales.

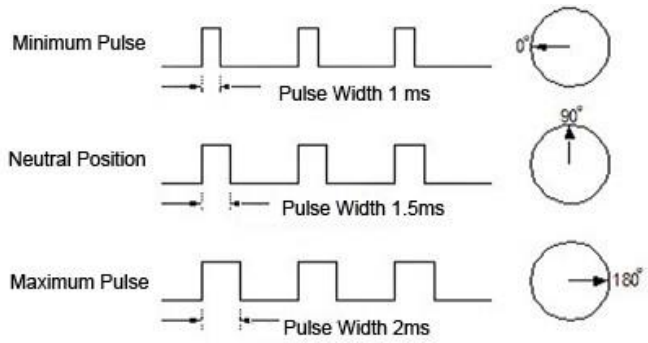

Imagen 4. PWM

5.2 PPM

La modulación por posición de pulso (PPM) (imagen 5) de sus siglas en inglés (*Pulse-Position Modulation*) está formada por N señales cuadradas donde la amplitud y el ancho de los pulsos son fijos, y la posición de estos variable.

En el caso del mundo del radiocontrol, las tramas PPM tienen una duración de aproximadamente 20 ms, donde cada uno de los pulsos que conforman la trama dura como máximo 2 ms. Este hecho nos permite deducir que el número de canales que se podrán multiplexar en una misma trama PPM es aproximadamente 10.

La ventaja que este tipo de modulación presenta es la disminución en el número de canales de transmisión, ya que las 4 señales PWM comentadas anteriormente se codificarían en una única trama PPM, requiriendo un único canal de Tx.

Para crear la trama PPM se requiere generar los pulsos de amplitud y ancho fijos en cada uno de los flancos de bajada la señal PWM a modular. El ancho de pulso es de aproximadamente 300 us.

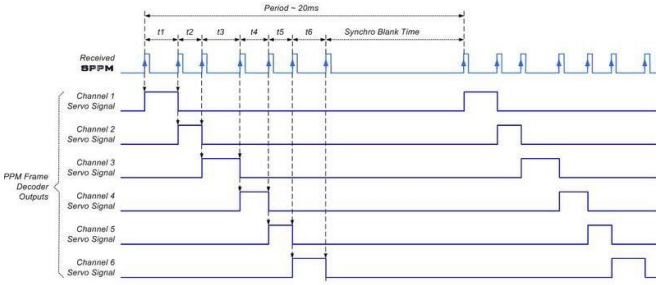

Imagen 5. PPM

6 IMPLEMENTACIÓN DEL SISTEMA

Podemos observar como se ha llevado a cabo la construcción de cada uno de los bloques presentados en la sección de "Sistema propuesto" en los apartados siguientes, donde se muestran las imágenes de las implementaciones.

Por otro lado, encontramos una explicación detallada del SW desarrollado para el bloque de gestión, donde encontramos comentadas las funciones implementadas.

6.1 Bloque de control

El bloque de control (imagen 6) ha sido implementado mediante el controlador DualShock 3, comentado en el apartado 4.1, y la placa USB_Host_Shield, comentada en el apartado 4.2.1. Observamos como la conexión se ha realizado mediante un cable miniUSB – USB.

Imagen 6. Bloque de control - Conexión mPS3-USB_Host_Shield

6.2 Bloque de gestión

El bloque de gestión comentado en el apartado 4.2 y que incluye el SW desarrollado y explicado en el apartado 4.2.2.1 mediante el diagrama de flujo de la ejecución (imagen 2), se ha implementado con el conjunto de funciones siguientes:

6.2.1 Setup

Observamos en la imagen 7 la implementación de la función de setup.

1. Esta zona del código se corresponde con la inicialización del timer de precisión y los puertos serie empleados para la transmisión del disparo y la obtención de las penalizaciones (Serial), y el testeo de errores (mySerial).
2. Esta zona se corresponde a la inicialización de las variables de control para la generación de la trama PPM.
3. Esta variable remarcada es la que gestiona el flujo de ejecución del programa. Permite dividir el flujo entre la gestión del mando y las penalizaciones, y la generación de la trama PPM.
4. Esta variable es la que permite tener el control sobre si la movilidad avanzada se encuentra activada o desactivada.
5. Esta zona inicializa las variables de tratamiento de mensajes del puerto serie.
6. Esta zona inicializa los valores PWM a modular mediante el protocolo PPM, así como las zonas de impacto del dron y su estado de no-penalizados.
7. Esta zona inicializa el pin digital 3, el cual se utiliza para la transmisión de la trama PPM.
8. Esta zona inicializa el USB para la gestión del mando de PlayStation 3.

```
480  void setup()
481  {
482    timer2.setup();
483    Serial.begin (115200);
484    mySerial.begin (115200);
485    matrix.begin(0x70);
486
487    lastFrLen =0;
488    lastServo =0;
489    lastPulse=0;
490    PPM_run =false;
491    pulse=false;
492    pulseStart = true;
493    counter =0;
494    part = true;
495
496    Cnt = 0;
497
498    ConAvanzado=false;
499
500    WeaponPaq = "RES";
501    ControllerPaq = "CONTROLLER1";
502
503    selectWeapon = 0;
504
```

```
505    for(int i=0; i<chanel_number; i++)
506    {
507      ppm[i] = default_servo_value;
508    }
509
510    for(int i=0;i<4;i++)
511    {
512        ShootZone[i]=false;
513        TagPaq[i] = "1";
514    }
515
516    pinMode(sigPin, OUTPUT);
517    digitalWrite(sigPin,onState);
518
519    #if !defined(__MIPSEL__)
520    while (!mySerial);
521    #endif
522
523    if (Usb.Init() == -1)
524    {
525      mySerial.print(F("\r\nOSC did not start"));
526      while (1);
527    }
528    mySerial.println(F("\r\nPS3 USB Library Started"));
529  }
```

Imagen 7. Función de Setup

6.2.2 Gestión del puerto serie

Observamos en las imágenes 8 y 9 la implementación de la gestión del puerto serie (lectura y decodificación).

1. Lectura: Esta función (PaRec) realiza una revisión del puerto serie en busca de datos que recoger. Durante el tiempo que hay datos disponibles se recogen y almacenan en un buffer. Una vez completada la tarea de recogida de datos, se realiza una conversión del buffer a una cadena de caracteres para su posterior decodificación.
2. Decodificación: Esta función (PaDec) en primer lugar realiza una comprobación de estructura para confirmar que el mensaje recibido es del tipo deseado (MOVEMENT). Posteriormente realiza la extracción de los 7 tags que contiene el mensaje, y que son los siguientes:
    a. Tag 1: MOVEMENT.
    b. Tag 2: Controlador asociado.
    c. Tag 3..6: Zonas penalizadas por impacto
    d. Tag 7: Tipo de Arma o Reset de las penalizaciones.

```
192  void PaRec()
193  {
194    if(Serial.available()) //Nos dice si hay datos dentro del buffer
195    {
196      //CTime(9,1);
197
198      memset(Paquete, 0,sizeof(Paquete));//memset borra el contenido del array  "cadena" desde la posición 0 hasta el final "sizeof"
199
200      while(Serial.available()>0) //Mientras haya datos en el buffer ejecuta la función
201      {
202        delay(5); //Poner un pequeño delay para mejorar la recepción de datos
203        Paquete[posicion]=Serial.read();//Lee un carácter del string "cadena" de la "posicion", luego lee el siguiente carácter con "posicion++"
204        posicion++;
205      }
206
207      PaqueteStr = String(Paquete);
208      posicion=0;//Ponemos la posicion a 0
209
210      PaDec();//Decodificación del mensaje
211    }
212  }
```

Imagen 8. Función de lectura del puerto serie

```
220  void PaDec()
221  {
222    int v=9;
223    int campos=0;
224
225    String aux = "";
226    PaqueteStr.toUpperCase();
227    /*
228      Estructura del paquete: TOPIC,CONTROLLER,IMP_AREA(1,1,1,1),WEAPON
229    */
230    if (PaqueteStr.startsWith("MOVEMENT,"))
231    {
232      while (campos<6)
233      {
234          while( (PaqueteStr[v] != ',') && (PaqueteStr[v] != '/') )
235          {
236            aux += PaqueteStr[v];
237            v++;
238          }
239
240          v++;
241
242          if(campos == 0)
243          {
244              ControllerPaq = aux;
245          }
246
247          if ( (campos>0) && (campos<5))
248          {
249              TagPaq[campos-1] = aux;
250          }
251
252          if(campos == 5)
253          {
254              WeaponPaq = aux;
255          }
256
257          aux = "";
258          campos++;
259      }
260    } else { Serial.println("ERROR: Estructura incorrecta");}
261  }
```

Imagen 9. Función de decodificación del mensaje

6.2.3 Gestión del tipo de arma

Podemos observar la implementación del segmento de código encargado de la gestión del tipo de arma en la imagen siguiente:

```
//Cambio de arma
if(PS3.getButtonClick(TRIANGLE))
{
    selectWeapon++;
    if ( selectWeapon >= 3) {
        selectWeapon = 0;
    }
}
```

Imagen 10. Función cambio de arma

El cambio de arma se realiza mediante la obtención del valor asociado al botón triangulo a través de la función "PS3.getButtonClick(TRIANGLE)" que gestiona el USB del DualShock. Cada vez que se recibe que el botón ha sido pulsado se actualiza la variable asociada al tipo de arma, la cual tiene 3 estados disponibles: "ICE", "ELE" y "NUC".

6.2.4 Gestión del disparo

La gestión de la orden de disparo y la transmisión del mensaje correspondiente se realiza mediante el código visible en las imágenes 11 y 12:

```
//Gestion disparo
if(PS3.getButtonClick(R1))
{
    PaSend();
}
```

Imagen 11. Gestión disparo

```
266  void PaSend()
267  {
268      if(Shoot==true)
269      {
270
271          if ( selectWeapon == 0) {
272              Serial.print("CFIRE,ice/");
273          }
274
275          if ( selectWeapon == 1) {
276              Serial.print("CFIRE,ele/");
277          }
278          if ( selectWeapon == 2) {
279              Serial.print("CFIRE,nuc/");
280          }
281      }
282  }
```

Imagen 12. Disparo

La gestión de la orden de disparo se realiza mediante la función "PS3.getButtonClick(R1)" que obtiene a través del USB el valor asociado al botón correspondiente. Posteriormente se realiza una llamada a la función "PaSend()" encargada de transmitir uno de los 3 tipos de mensaje de disparo por puerto serie (pin digital 1) en función del tipo de arma equipada.

6.2.5 Gestión de movilidad avanzada

La activación o desactivación de la movilidad avanzada se consigue con el fragmento de código de la imagen 13, donde observamos que se emplea la función "PS3.getButtonClick(L1)" para comprobar que el botón asociado ha sido pulsado.

```
//Gestion complejidad maniobras
if(PS3.getButtonClick(L1))
{
    ConAvanzado =! ConAvanzado;
}
```

Imagen 13. Gestión movilidad avanzada

Por otro lado, la gestión de la movilidad avanzada se realiza en el fragmento de código de la imagen 14, donde observamos que se modifica el valor PWM de dos canales simultáneos y correspondientes al Yaw y Roll.

```
//Desplazamiento complejo
if((PS3.getAnalogHat(RightHatX) > -1) && ConAvanzado)
{
    ppm[0] = constrain(PS3.getAnalogHat(RightHatX), 10, 245);
    ppm[0] = map(ppm[0], 10, 245, 1100,1900);

    ppm[4] = constrain(PS3.getAnalogHat(RightHatX), 10, 245);
    ppm[4] = map(ppm[4], 10, 245, 1100,1900);
}
```

Imagen 14. Movilidad avanzada

6.2.6 Gestión de vuelo

La gestión de vuelo se realiza mediante la sección de código de la imagen 15, donde apreciamos que se comprueba si los botones asociados al movimiento del dron (explicados en la sección 4.1) se han utilizado. En el caso de que se haya utilizado alguno de los botones, se obtiene el valor producido y se realiza una comprobación y una transformación. En primer lugar, la comprobación realizada mediante la función "constrain(…)" nos asegura de que los valores recibidos están comprendidos en el rango de valores esperado. En segundo lugar, la transformación realizada mediante la función "map(…)" nos genera un valor de PWM en función del dato de entrada generado por el mando. Cada uno de los valores PWM se corresponde a un canal.

```
//Throttle
if(PS3.getAnalogButton(R2) > -1)
{
    ppm[2] = constrain(PS3.getAnalogButton(R2), 10, 245);
    ppm[2] = map(ppm[2], 10, 245, 1100,1900);
}

//Pitch
if(PS3.getAnalogHat(RightHatY) > -1)
{
    ppm[1] = constrain(PS3.getAnalogHat(RightHatY), 10, 245);
    ppm[1] = map(ppm[1], 10, 245, 1900,1100);
}

//Yaw
if((PS3.getAnalogHat(RightHatX) > -1) && !ConAvanzado)
{
    ppm[0] = constrain(PS3.getAnalogHat(RightHatX), 10, 245);
    ppm[0] = map(ppm[0], 10, 245, 1100,1900);
}

//Roll
if((PS3.getAnalogHat(LeftHatX) > -1) && !ConAvanzado)
{
    ppm[4] = constrain(PS3.getAnalogHat(LeftHatX), 10, 245);
    ppm[4] = map(ppm[4], 10, 245, 1100,1900);
}
```

Imagen 15. Gestión de vuelo

6.2.7 Gestión de las penalizaciones

Las penalizaciones se gestionan en dos fases diferenciadas y visibles en las imágenes 16 y 17.

La primera fase, correspondiente a la función "SimShot()", asigna a cada parte del dron el estado de penalización (true o false) obtenido de la decodificación del mensaje de "MOVEMENT" en función del tipo de arma o de si es un reset (Tag PENALTY).

La segunda fase, mediante la función "constrain (…)", ajusta el valor de PWM de la trama PPM a un rango inferior al utilizado por defecto ([1100,1900]). Esta reducción del rango de valores reduce la potencia transmitida al motor (sección 5.1).

Una vez terminada la gestión de las penalizaciones el flujo del programa se redirecciona a la generación de la trama PPM.

```
void SimShot()
{
    if (WeaponPaq.equals("ICE"))
    {
      if(TagPaq[2].equals("0")) {ShootZone[0] = true;}//ALANTE

      if(TagPaq[1].equals("0")) {ShootZone[2] = true;}//iZQUIERDA

      if(TagPaq[0].equals("0")) {ShootZone[1] = true;}//DERECHA

      if(TagPaq[3].equals("0")) {ShootZone[3] = true;}//ATRAS

    }

    if (WeaponPaq.equals("ELE"))
    {
      Shoot = false;
    }

    if (WeaponPaq.equals("NUC"))
    {
        ShootZone[0] = true;
        ShootZone[1] = true;
        ShootZone[2] = true;
        ShootZone[3] = true;
    }

    if (WeaponPaq.equals("PENALTY"))
    {

        if(TagPaq[2].equals("1")) {ShootZone[0] = false;}//ALANTE

        if(TagPaq[1].equals("1")) {ShootZone[2] = false;}//iZQUIERDA

        if(TagPaq[0].equals("1")) {ShootZone[1] = false;}//DERECHA

        if(TagPaq[3].equals("1")) {ShootZone[3] = false;}//ATRAS

        Shoot = true;
```

Imagen 16. Gestión impacto

```
void EfectosMovilidad()
{
    if(ShootZone[0]==true)//Reduce Pitch.
    {
        ppm[1] = constrain(ppm[1], 1400, 1600);
    }

    if((ShootZone[1]==true))//Reduce Yaw derecha
    {
        ppm[0] = constrain(ppm[0], 1100, 1600);
    }

    if((ShootZone[2]==true))//Reduce Yaw izquierda
    {
        ppm[0] = constrain(ppm[0], 1400, 1900);
    }

    if(ShootZone[3]==true)//Reduce Roll
    {
        ppm[4] = constrain(ppm[4], 1400, 1600);
    }

    if((ShootZone[0]==true) && (ShootZone[1]==true) && (ShootZone[2]==true) && (ShootZone[3]==true))//Reduce Throttle
    {
        ppm[2] = constrain(ppm[2], 1100, 1500);
    }
}
```

Imagen 17. Gestión penalizaciones

6.2.8 Creación de la trama PPM

La generación de la trama PPM a partir de los valores PWM generados tras la gestión de vuelo y la aplicación de las penalizaciones se realiza en la función mostrada en la imagen 18.

```
void ppmWrite()
{
  if(timer2.get_micros() - lastFrLen >= PPM_FrLen)//start PPM signal
  {
    lastFrLen = timer2.get_micros();
    PPM_run = true;
    timePPMIni = timer2.get_micros();
  }

  if(counter >= chanel_number)
  {
    PPM_run = false;
    counter = 0;
  }

  if(PPM_run)
  {
    if (part)
    {
        pulse = true;
        part = false;
        lastServo = timer2.get_micros();
    }
    else
    {
        if(timer2.get_micros() - lastServo >= ppm[counter])
        {
            counter++;
            part = true;

            if(counter == chanel_number)
            {
                digitalWrite(sigPin, !onState);
                float lim = timer2.get_micros();

                while(timer2.get_micros() - lim >= PPM_PulseLen)
                {}

                digitalWrite(sigPin, onState);
                Cnt=0;
            }
        }
    }
  }

  if(pulse)
  {
    if(pulseStart == true)
    {
        digitalWrite(sigPin, !onState);

        pulseStart = false;
        lastPulse = timer2.get_micros();
    }
    else
    {
      if(timer2.get_micros() - lastPulse >= PPM_PulseLen)
      {
                digitalWrite(sigPin, onState);

        pulse = false;
        pulseStart = true;
      }
    }
  }
}
```

Imagen 18. Creación trama PPM

Podemos observar en el código como la primera sección se corresponde con la gestión del inicio de la trama PPM, donde se determina que cada 20 ms comienza una nueva trama. Posteriormente se realiza la gestión del canal a modular y se acciona un pulso.

Una vez alcanzado el periodo de la señal PWM se cambia de canal y se acciona nuevamente un pulso, situando así cada pulso en la posición correcta en la trama.

El resultado es una trama con un número de pulsos igual al número de canales, y donde la posición de cada pulso varía en función del ancho del PWM asociado al canal.

La generación de la trama PPM se genera a tiempo real a lo largo de los 20 ms a través del pin digital 3, donde se aplican los cambios de estado de los pulsos (high/low).

Una vez terminada la trama PPM el flujo del programa se redirecciona a la gestión del vuelo y las penalizaciones.

6.3 Bloque de simulación

El bloque de simulación está constituido por el simulador Liftoff comentado en el apartado 4.3, el cual se sincroniza con el módulo de gestión mediante el pin digital 3 (Imagen 19).

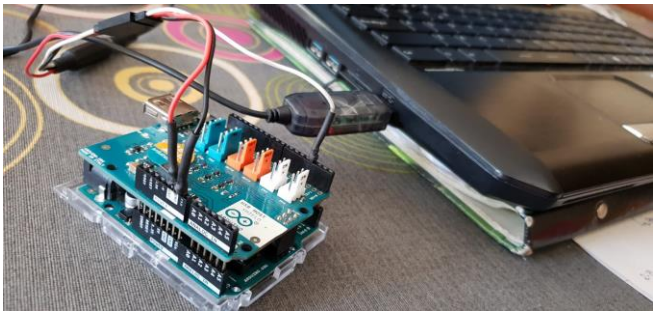

Imagen 19. Conexión Arduino Simulador

En la imagen 20 podemos observar la página principal del simulador, donde encontramos los apartados más relevantes:

- *Single Player*: En esta categoría del menú accedemos al modo "Un jugador". Este modo de juego ha sido el empleado para testear el correcto funcionamiento del sistema.
- *Multiplayer:* Esta categoría del menú da acceso al modo "Multijugador", y ha sido empleada para testear la jugabilidad conseguida con el sistema.
- *Options:* Esta sección del menú da acceso a la configuración de los controles, y ha sido utilizada para sincronizar el sistema con el simulador mediante el protocolo PPM.

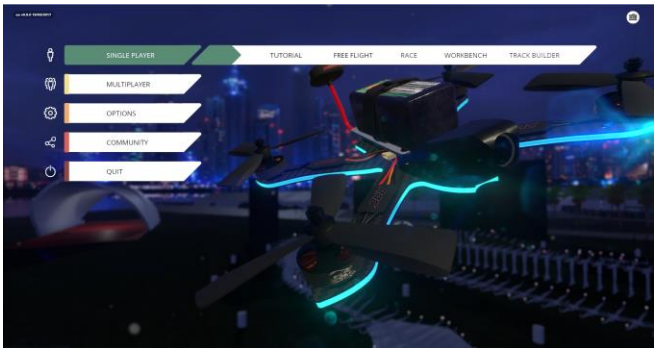

Imagen 20 .Página principal Liftoff

En la imagen 21 nos encontramos con el panel de selección de aeronave, dentro del modo "Un jugador", donde observamos que existe una amplia variedad de drones.

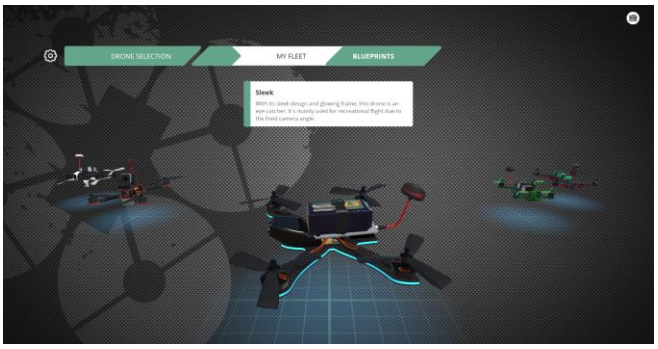

Imagen 21. Selección de dron - Liftoff

Posteriormente, en la imagen 22, visualizamos el escenario de simulación de vuelo libre, y en la imagen 23 el entorno de carrera. Estos escenarios son unos de los varios disponibles dentro del entorno de simulación.

Imagen 22. FreeFly - Liftoff

Imagen 23. Race - Liftoff

La imagen 24 muestra el entorno de configuración de los controladores. Este es el modelo que sigue cada una de las pantallas asociadas a los diferentes botones del controlador.

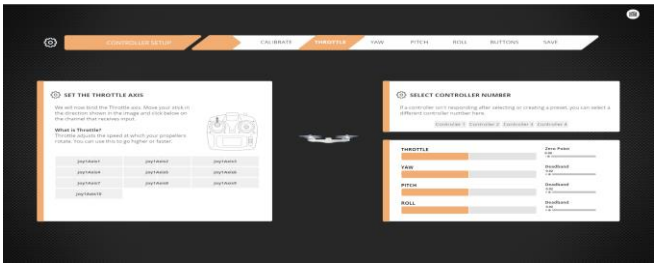

Imagen 24. Configuración del controlador mediante PPM - Liftoff

6.4 Bloque de transmisión y recepción

El bloque de transmisión y recepción está compuesto por el mando de dron y el receptor explicados en el apartado 4.4. La conexión con el módulo de gestión se ha realizado mediante un cable con clavija mini-Din de 4 pines.

7 METODOLOGÍA

La metodología de trabajo empleada a lo largo del curso para el desarrollo del proyecto "Módulo PSxDron de gestión de vuelo y penalizaciones por impacto IR" se ha basado en la metodología SRUM. A lo largo del desarrollo se han ido planificando puntos de control semanales tutorizados para la comprobación de los objetivos marcados, así como para el planteamiento de nuevos objetivos.

El conjunto de sprints realizados forman parte de cada una de las fases explicadas a continuación:

- **Fase I:** Investigación acerca de la gestión del mando de PS3 con la placa Arduino.
- **Fase II:** Investigación acerca de la implementación del protocolo PPM.
- **Fase III:** Desarrollo de un código de prueba para la gestión del mando de PS3.
- **Fase IV:** Testeo del código de prueba de la gestión del mando y corrección de errores.
- **Fase V:** Desarrollo de un código de prueba para la implementación del protocolo PPM.
- **Fase VI:** Testeo del código de prueba para la generación de las tramas PPM y corrección de errores
- **Fase VII:** Desarrollo de un código de prueba para la traducción de las ordenes de movimiento a un valor compatible con el mando de dron.
- **Fase VIII:** Testeo del código de traducción de órdenes y corrección de errores.
- **Fase IX:** Integración del código de gestión del mando de PS3 con la generación de tramas PPM y la traducción de órdenes. Generamos la primera versión "completa" del SW (V1).
- **Fase X:** Testeo del SW en el simulador de vuelo y corrección de errores.
- **Fase XI:** Desarrollo de un código de prueba para la gestión de las penalizaciones de movilidad.
- **Fase XII:** Testeo del código de gestión de penalizaciones y corrección de errores.
- **Fase XIII:** Integración del código de gestión de penalización en la versión 1 del SW. (Generamos la versión final)
- **Fase XIV:** Testeo de la versión final del SW en el simulador de vuelo y corrección de errores.

Las fases comentadas anteriormente se llevaron a cabo de la siguiente manera:

- **Fase I y II:** Investigación acerca de las librerías existentes para dicho propósito. Finalmente la librería escogida es la USB_Host_Shield para el caso de uso PS3, debido a la buena valoración de la comunidad de desarrolladores, y la incorporación de códigos de ejemplo que facilitan la com-

prensión y adaptación del programa. Una vez hecha la elección se utilizó el conjunto de funciones de la librería para realizar un programa de gestión del mando que permitiera la emisión de las órdenes de movimiento a la placa Arduino.

- **Fase III y V:** Investigación acerca de códigos de ejemplo para generar tramas PPM. Una vez obtenido un conjunto de ejemplos significativo y contrastado se eligió el más amigable y fácil de entender. Posteriormente se realizó una adaptación del mismo que permitiera la correcta generación de la trama PPM.
- **Fase IV y VI:** Una vez realizados los programas de gestión del mando y generación de la trama PPM, se diseñaron unos sencillos ficheros de prueba que permitieran visualizar el correcto funcionamiento por pantalla. Para el caso de la gestión del mando, el programa de prueba permitía visualizar un valor variable en función de la posición del joystick comprendido entre [0,255]. Para el caso de la trama PPM se realizó un programa que cambiaba los valores de PWM y que alteraba los anchos de pulso de la trama PPM. Dichos cambios en los anchos de pulso podían observarse por pantalla. Adicionalmente se comparó la trama generada por la placa Arduino con la generada por un mando comercial de dron mediante un osciloscopio.
- **Fase VII, VIII, IX y X:** Una vez contamos con los módulos independientes de gestión del mando de PS3 y la generación de tramas PPM, se integraron dichos módulos de manera que los valores obtenidos (de 0 a 255) de las órdenes del mando se convirtieran en valores compatibles con la trama PPM (de 1000 a 2000). Para ello se generó una lógica de traducción que realizara la conversión de los valores de [0,255] a valores comprendidos entre [1000,2000]. Posteriormente se testeo el funcionamiento mediante el uso de un simulador de vuelo, observando que podía controlarse el dron. (Programa obtenido: Versión 1 del SW)
- **Fase XI y XII:** Diseño del tipo de armas disponibles así como de los efectos que generan con los impactos. Se establecen 3 tipos de armas con los efectos correspondientes:
    - **"Hielo":** Al impactar ralentiza la zona de impacto, reduciendo la movilidad del dron.
    - **"Eléctrica":** Imposibilita la acción de disparar.
    - **"Nuclear":** Reduce la movilidad del dron así como la potencia de vuelo.

  Posteriormente se genera un módulo de gestión de penalizaciones donde se realiza la descodificación de los paquetes recibidos indicando impactos, así como la aplicación del efecto correspondiente.

  Para el testeo se realizan envíos por puerto serie siguiendo el protocolo acordado, comprobando si el módulo realiza una correcta descodificación así como una correcta aplicación de los efectos. El resultado de los efectos puede comprobarse mediante la visualización de los valores de PWM por pantalla y osciloscopio.

  Se ha creído conveniente como extra, generar un módulo para la visualización del tipo de arma mediante un display que permite llevar un control del efecto esperado.

  El display utilizado es un 7-segmentos que se comunica con la placa arduino mediante un bus I2C. Este bus de comunicación cuenta con 4 líneas de conexión:
    - SDA: Línea de datos
    - SCL: Línea de clock
    - Vdd: Línea de alimentación:
    - GND: Línea de tierra
- **Fase XIII y XIV:** Una vez generado el módulo de gestión de penalizaciones se integra junto a la versión 1 del SW PSxDron. Esta nueva versión (v2.0) que incluye la gestión del mando de PS3, la generación de la trama PPM y la gestión de las penalizaciones, permite controlar un dron sensible a impactos por arma congelante, eléctrica y nuclear, y visualizar los efectos en el simulador. Este hecho es el utilizado para el testeo de la versión 2.0.

En cuanto a la metodología de desarrollo, esta se ha llevado a cabo mediante las herramientas siguientes:

- **VisualStudio:** Entorno de desarrollo integrado de Microsoft empleado para la creación del SW del módulo PSxDron.
- **IDE Arduino:** Entorno de desarrollo de Arduino empleado mayormente para la subida del código a la placa Arduino UNO, así como para la visualización de datos por el puerto serie.
- **Hércules:** Terminal de puerto serie empleado para el uso de un segundo puerto serie de simulación de penalizaciones.
- **Liftoff:** Simulador empleado para testear el funcionamiento del módulo PSxDron en un entorno virtual.

8 RESULTADOS

A lo largo del desarrollo del proyecto se han ido encontrando diversos problemas que se han tenido que analizar, resolver y en algunos casos mitigar.

Los problemas encontrados más relevantes y las medidas tomadas, así como los resultados obtenidos, se detallan a continuación:

- Una vez completada las fases de desarrollo y testeo de la gestión del mando de PS3 (fases III, VII y fases IV, VIII), y

de generación y testeo de la trama PPM (fase V y fase VI), se realiza la fase IX de integración de ambos módulos y la fase X de testeo del resultado en el simulador. Pese a que ambos módulos por separado producían resultados correctos, al realizar la integración los resultados son los siguientes:
    - El simulador no sincroniza correctamente el módulo.

  En esta situación se recurre a analizar mediante un osciloscopio la trama PPM generada tras la integración, encontrando que esta no se genera correctamente. La conclusión a la que se llega es que la gestión del mando produce un retraso en la generación de la trama PPM, por lo que la medida tomada es la siguiente:
    - Gestionar el flujo de ejecución del programa mediante un IF ELSE. Primero se gestiona el mando y posteriormente se crea la trama PPM con los valores obtenidos de la gestión.

  Tras aplicar la medida se analiza la trama y se comprueba la correcta generación de la misma.
- Una vez bifurcado y controlado el flujo de ejecución se realiza nuevamente la fase X, obteniendo los resultados siguientes:
    - El simulador nuevamente no sincroniza correctamente el módulo.

  En esta situación se recurre a analizar la trama generada por un mando de dron comercial, obteniendo los siguientes resultados:
    - La trama generada por el módulo PSxDron es parecida a la generada por el mando de dron comercial. La diferencia reside en el hecho de que están invertidas.
    - Los tiempos de duración de ambas tramas no se corresponden, aunque la diferencia de tiempos es pequeña.

  En esta situación se aplican las medidas siguientes:
    - Inversión de la trama PPM generada para hacerla igual a la del dron comercial, ya que la de este si sincroniza con el simulador.
- Una vez invertida la trama PPM se realiza nuevamente la fase X, obtenido los siguientes resultados:
    - El simulador sincroniza con el módulo PSxDron y nos permite controlar el dron.
    - La estabilidad del dron no es buena, por lo que el requisito de jugabilidad se ve comprometido

  Las posibles causas de la inestabilidad a las que se llega son la de irregularidades en la generación de la trama PPM por falta de precisión en el clock de la placa, o posibles interferencias en la señal producida por el mando de PS3. Las medidas que se toman son las siguientes:
    - Investigación acerca de cambios en el clock de la placa que solucionen o mitiguen el problema, e implementación de la solución
- Una vez encontrada una librería que aumentaba la precisión del clock se realiza la fase X, obtenido los resultados siguientes:
    - La inestabilidad del dron se ha corregido en gran parte, aunque sigue quedando una pequeña interferencia residual en la movilidad. Pese a ello se considera que la jugabilidad para un prototipo es adecuada.

Una vez finalizadas el resto de fases se trató de mitigar las pequeñas interferencias residuales visibles en la movilidad del dron mediante las medidas siguientes:

- Función de promediado: Esta función realiza un promediado de la señal generada por el mando de PS3 y se aplica previamente a la generación de la trama PPM a modo de filtro. El promediado consiste en la gestión de un buffer donde se almacenan los diversos valores de movimiento que se van obteniendo, y el resultado esperado es mitigar los valores anómalos interferentes. Es decir, si el valor PWM que se quiere aplicar es 1700, y se reciben pequeños picos (1710 por ejemplo), estos se verán eliminados por el promedio de todos los valores almacenados. Los resultados obtenidos tras la implementación e integración de la función son los siguientes:
    - Las interferencias residuales en la movilidad del dron no se ven reducidas ni aumentadas.
    - La movilidad del dron se ve ralentizada al aplicar un cambio brusco en alguno de los movimientos debido al tiempo que conlleva actualizar el buffer de promediado.
- Función mapExponencial: El objetivo de esta función era substituir la función de transformación map(), la cual realiza una conversión lineal de un rango de entrada a uno de salida, por una función cuyo cometido es el mismo pero mediante una transformación exponencial. El resultado esperado era la mitigación del ruido en los valores cercanos a 1500 (valor de parada de los servomotores). El resultado obtenido tras la implementación y aplicación de la función es el siguiente:
    - Las interferencias residuales en la movilidad del dron no se han visto alteradas.

Los resultados alcanzados tras el desarrollo del proyecto son los siguientes:

- Controlador de vuelo de un cuadricóptero mediante el uso de un mando de PS3.
- Gestión de la orden de disparo y transmisión de la misma por puerto serie.

- Gestión del tipo de arma equipada.
- Gestión de los impactos recibidos de manera que se vean los efectos aplicados.
- Tiempo de latencia entre que se genera la orden de disparo y esta se transmite por el puerto serie 2.09 ms

Además, en una fase inicial de desarrollo, se cuenta con un módulo de visualización para la cantidad de vidas de las que se dispone y el tipo de arma que se lleva equipada.

9 Conclusiones

Una vez alcanzado el plazo de tiempo asignado para el desarrollo del trabajo de final de grado podemos concluir que se han logrado los objetivos previamente marcados en la etapa de gestión y definición del proyecto. Esto es debido a que el estado en el que se encuentra el proyecto de diseño y desarrollo del módulo PSxDron de gestión de vuelo y penalizaciones engloba las siguientes características:

- Gestión del mando DualShock 3 (PS3).
- Gestión de las penalizaciones existentes en el modo de juego *Game of Drones.*
- Gestión de la orden de disparo.
- Tiempo de latencia de 2.09 ms

Todas estas características se consiguen respetando los requisitos del sistema, en cuanto a tiempo de respuesta y jugabilidad, establecidos para el prototipo.

10 Consideraciones futuras

Las siguientes fases de desarrollo que se podrían llevar a cabo en el proyecto son las siguientes:

- Conexión del mando DualShock 3 con el módulo PSxDron mediante Bluetooth.
- Efecto de vibración al recibir un impacto en el dron.
- Ampliar la movilidad avanzada incorporando la detección de movimiento mediante acelerómetro del DualShock 3.
- Añadir otras funcionalidades de visualización en el display como el tiempo de penalización, la zona afectada y el nivel de efecto.
- Refinar la estabilidad en la movilidad del dron.

Agradecimientos

Esta sección es para dar las gracias en primer lugar a Màrius Montón, tutor del trabajo de final de grado, y a David Matanzas, colaborador de AIRK Drones, por la paciencia y el asesoramiento a lo largo del desarrollo del trabajo de final de grado, así como por la implicación que han demostrado a lo largo de este tiempo.

En segundo lugar, pero no menos importante, a mis compañeros Ander Pardo, Iván Yera y Marc Guerrero por su disposición y ganas de afrontar nuevos retos juntos, y sin los cuales esto no habría sido posible.

Bibliografía

[1] David, Guía de manejo de drones para principiantes. MiniDrones.
[2] Arduino USB Host-Shield.
[3] Como funcionan y vuelan los drones. 11 de Junio del 2017
[4] Enrique Gómez, PWM
[5] FPVMAX, Protocolos de comunicación R/C
[6] Gabriel Staples, Timer de 0.5 us de precisión. 9 de Febrero del 2014
[7] Jim Eli, Timer Arduino. 17 de Marzo del 2012
[8] Oleg_Mazurov,_Alexei_Glushchenko,_Kristian_Lauszus,_Andrew Kroll, Guruthree, Yuuichi Akagawa, USB_HostShield_2.0.